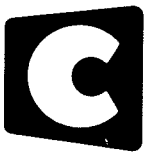
Five Dollars



Cromemco

Link & Lib

Instruction Manual

Cromemco™

LINK and LIB

REFERENCE MANUAL

CROMEMCO, Inc.
280 Bernardo Avenue
Mountain View, CA 94043

Part no. 023-4026

September 1980

Copyright © 1980
By CROMEMCO, Inc.
ALL RIGHTS RESERVED

This manual was produced on a Cromemco System Three computer using the Cromemco Word Processing System. Final camera-ready copy was printed on a Cromemco Model 3355A printer.

CROMEMCO LINK and LIB

Reference Manual

Table of Contents

INTRODUCTION

The Cromemco Link and Lib Manual consists of two chapters describing the use of two frequently-used utility programs. The first program is Link, a program which transforms the relocatable files output from the Cromemco Assembler and the C, Cobol, Fortran, and RPG II compilers into executable form. Link is supplied with each of the above software packages.

The second program is Lib, which can be used to concatenate multiple relocatable files into a single library which the Link program can search while linking, for example, the relocatable output of the C compiler. Lib also can list all symbols defined within a relocatable file, and generate a cross-reference of all symbols defined within a relocatable library.

SECTION 1

Link

Link is the program which loads relocatable object files into memory and changes all relative addresses into actual memory addresses. Link can load one or more files during one execution and search a user-specified library of relocatable routines to load commonly-used modules of code. The typical high-level language user will use Link to load one file, output from one of the Cromemco language compilers, and to search the relocatable library supplied with the compiler for commonly used routines.

The typical assembly-language user will first assemble multiple source files to produce multiple relocatable files, then use Link to load all of these modules into one executable file.

Throughout this manual, the word **module** refers to the relocatable file assembled or compiled from one source file. A **library** is a file which contains one or more modules concatenated together. A **global symbol** is a symbol, or name, of either a data location or a program routine which is accessible from other modules than the one in which the symbol is defined.

1.1 Link Commands

Link can be executed in both command line mode and in prompt mode. For simple linking tasks, the command line mode is more efficient, and is the mode typically used to link C, Cobol, Fortran, and RPG II programs. In this mode, the user types **Link** followed by the name of the program to link.

Example:

```
link program,program/n/e
```

This will cause Link to load the relocatable file named **program.rel** from the current disk, search the default library for any undefined symbols, create a new executable file with the name **program.com** on the current disk, and return to CDOS. The name of the default library has been passed to Link by the compiler which compiled **program.** (See special Link item 3, discussed in 1.7, Relocatable File Format.)

Prompt mode

The prompt mode is typically used when more modules are to be linked than can be typed on one command line. To execute Link in this mode, type **Link** followed by a carriage return. Link will respond with the prompt character, *, indicating that it is ready to accept commands.

A command to Link consists of a list of file names and command switches. Successive file name-switch groupings must be separated by commas:

```
d:fnamel.ext/s, d:fname2.ext/s, ...
```

where **d:** is the letter of the drive on which the file is located, **fnamel.ext** is the file name, and **s** is the letter of one command switch.

d: is optional; the default is the letter of the currently logged disk.

.ext is optional; the default varies depending on the current command.

After each line is typed, Link will load or search (see the **/s** switch below) the specified files. After Link loads all the files, it will list all symbols that remain undefined, followed by asterisks.

Example:

```
*ddgo

Data    0103    0136

GNUMB.W*        011B    MENU*   0133    TOPLINE*    012D
TQITERA*        0125    GOML1*  0118    GOML2*  0130
GOM1*   012A
7 Undefined Global(s)
```

1.2 Link Command Switches

Link recognizes several command switches that can be used to effect more precise control or to provide more information during the linking process. Each switch must be preceded by a slash, /.

Switches can be typed in various places in the command, depending on the switch and the intended effect of the switch.

1. A switch can be the entire command, or can appear first in the command.

2. A switch can appear immediately following a file name. **/s** and **/n** apply to only one file name in the command, so when either is used, it is placed immediately after the file name.

3. Any switches that affect the entire Link session may appear at the end of the command.

The descriptions of the available switches follow.

Switch Action

D **/d:nnnn** causes Link to set the **data** origin to address nnnn, expressed in the current display radix. The data origin is set for the next module loaded--previously loaded modules are not affected.

Do not use **/d** to set the data origin within the address range 100H-102H. Link will not insert the jump instruction at 100H to the start-execution address if the data area has been loaded there.

A complete discussion of the interactions of the **/d** and **/p** switches and the final memory locations of data, program, and common areas can be found in 1.4, Data, Program, and Common Areas.

E or **E:name**

/e will cause Link to **exit** and return control to the operating system. Link will first search the default library (if one has been requested by the language compiler) in an attempt to find and load modules containing the definitions of any undefined globals.

The form **/e:name**, where **name** is a global symbol defined in one of the loaded modules, will cause Link to change the destination of the jump instruction at 100H (which is inserted by Link) to the memory location defined by **name** before exiting. Execution of the new program will therefore start at the selected location whenever the new program is executed.

G or **G:name**

/g will cause Link to start execution of the program (**go**) as soon as the current line of commands has been completely interpreted. Link will first search the default library for undefined globals if such a library has been requested by the language compiler. Then Link will display three numbers and the BEGIN EXECUTION message. The three numbers are the start-execution address of the program, the address of the first available byte of memory following the program, and the number of 256-byte pages of memory used.

The form **g:name**, where **name** is a global symbol defined in one of the loaded modules, starts execution at the program location defined by **name.**

H **/h** sets the radix of all displayed numbers to **hexadecimal.** This is the default.

M **/m** causes Link to list the starting and ending addresses of the program and data areas, and a **map** of all global symbols, whether defined or undefined. The defined symbols are listed with their values; each undefined symbol is listed with an asterisk following the last letter of the symbol.

The starting and ending addresses of the program area are displayed only if **/d** has been used to separate the data and program areas. Otherwise, the program area is stored as part of the data area.

N **filename/n** causes Link to create a **new** executable file with the name **filename.com** when **/e** or **/g** is typed. If necessary, Link will insert a jump instruction at 100H to the start of the program so that the program will execute correctly when loaded at 100H by the operating system.

O **/o** sets the radix of all displayed numbers to **octal.** Hexadecimal is the default.

P **/p:nnnn** causes Link to set the **program** origin

to address nnnn, expressed in the current display radix. The program origin is set for the next module loaded--previously loaded modules are not affected. The default program origin is 103H, to leave room for the jump to the program's start address. Do not use **/p** to set the program origin within the address range 100H-102H, unless it is to load the start of the program there. Link will not insert the jump instruction at 100H if a data or program area has been loaded there.

See 1.3, Starting Program Address, for more details about the starting program address.

See 1.4, Data, Program, and Common Areas, for a complete discussion of the interactions of the **/d** and **/p** switches and the final memory locations of data, program, and common areas.

R **/r** causes Link to **reset** to its initial state. Link resets immediately when it reads **/r** from the command line. **/r** can be used to restart Link after a wrong file has been inadvertently loaded.

S **/s** will cause Link to **search** the file whose name precedes the **/s** to satisfy any undefined globals. The file is usually a library of relocatable modules. When Link finds a definition for a currently undefined global, it will load the entire module from the library.

The C, Cobol, Fortran, and RPG II compilers all use Link item 3 to signal Link to search the appropriate library automatically.

Note that Link does not display any still-undefined globals after a library search; the user must use **/u** after **/s** to learn whether any globals remain undefined.

U **/u** causes Link to list the starting and ending addresses of the program and data areas, and a list of all **undefined** global symbols. Link will display this list after it interprets the rest of the command line, although **/u** is typically used by itself.

The starting and ending addresses of the

program area are displayed only if **/d** has been used to separate the data and program areas. Otherwise, the program area is stored as part of the data area.

X **/x** is used in conjunction with **/n** and **/e** to cause Link to create a file in the Intel ASCII HEX format. Link creates the file, named **filename.hex**, instead of the executable file named **filename.com**. The correct usage is **filename/n/x/e.**

See 1.8, Hex File Format, for a description of the hex format.

Y **/y** is used in conjunction with **/n** and **/e** to cause Link to create an additional file, named **filename.sym**, when **/e** is typed. This file will contain the names and memory addresses of all globals, information which is essential when debugging programs. The correct command usage is **filename/n/y/e.**

Examples:

*/g Causes Link to pass control to the start of the previously loaded program.

*/m Causes Link to list all global symbols defined in the previously loaded modules.

*progl,progl/n/e

Causes Link to load **progl.rel**, create a new file named **progl.com** on disk, and return control to the operating system.

*dump/n,dump/m/e

Causes Link to create an executable file named **dump.com**, load **dump.rel**, list all global symbols, save the executable file, and return control to the operating system.

*/p:200,exampl

Causes Link to load **exampl.rel** so that the

program area starts at address 200.

*usrlib/s,userprog

Causes Link to search **usrlib.rel** and load any necessary modules, then load **userprog.rel.**

*prog1/n/y/e

Causes Link to create two files: **prog1.com** and **prog1.sym.**

*prog2/n/x/y/e

Causes Link to create two files: **prog2.hex** and **prog2.sym.**

1.3 Starting Program Address

When a user requests that the operating system execute a program, the system will load the program into memory starting at address 100H, and pass control to the instruction at 100H. Execution of a program which contains the data area before the program area will start as usual at 100H. No one desires that execution begin in the data area, so a method has been developed to cause execution to begin at a user-selected start address within the program. The start address is specified either by the **end <symbol>** line at the end of the assembler source file, or by the **/e:name** Link command. **<symbol>** in the first case is a label used within an assembly language source module, usually the main module of the program; **name** in the second case is a global symbol defined in any of the loaded modules.

Link will insert a jump instruction at 100H to transfer control to the start address when one has been specified. When no start address is specified, Link will insert three bytes of 0 (Z80 nop instructions) at 100H. Link will not insert the jump or zero bytes if either **/d** or **/p** has been used to set the data or program origin within the memory range 100-102H.

The assembly language programmer must specify the start address for the program. This is usually done by choosing one module as the main module, declaring a label named **<symbol>** before the first opcode, and using the **end <symbol>** line to end the module. Note that only one module can be so selected in a program.

The C, Cobol, Fortran, and RPG II libraries each contain a main module, so the user of one of these languages need not specify a start address. The user does need to make certain that an assembly language subroutine called from one of these high-level languages does not contain an **end <symbol>** line.

1.4 Data, Program, and Common Areas

An executable program can contain three types of areas: data areas, program areas, and any number of common areas. A data area contains variables and data storage locations used by one module of the program to perform its tasks; a program area contains the program code which performs the tasks; and a common area is a data area which can be shared among the several program modules which may comprise the executable program. A common area can be identified with a name 1-6 characters long or it can have no name, in which case it is called blank common. Link loads common areas with the same name starting at the same location in memory, thereby effecting the overlay property of common areas.

A relocatable module output from the Cromemco Macro Assembler or one of the C, Cobol, Fortran, or RPG II compilers can contain one or more of each type of area, depending on the programmer's methodology and the language used.

Link does not initialize data or common areas--that task is left to the programmer. Data values in a common block can be defined in Fortran and Assembler programs, and the final values in a common block referenced in more than one loaded module will be the values in the common block in the last module loaded.

Within the individual modules, a common area identified with the same name can have different lengths, but the module containing the largest definition of that common area must be loaded before other modules which contain that common area.

The final memory location of each program, data, and common area loaded depends on the use of the **/d** and **/p** switches, as described in the following paragraphs. Remember that common areas with the same name will start at the same location.

Default locations:

When neither **/d** nor **/p** is used, Link considers the program area to be part of the data area and loads all areas from all modules so that the areas have the

following order in memory, starting at 103H: all common, data, and program areas from the first module, followed by all common, data, and program areas from the second module, and so on for each module loaded.

/d only:

When only **/d** is used, Link separates the program areas from the data and common areas so that the program areas have the following order in memory, starting at 103H: all program areas from the first module, followed by all program areas from the second module, and so on for each module loaded.

The data and common areas have the following order in memory, starting at the specified data origin: all common and data areas from the first module, followed by all common and data areas from the second module, and so on for each module loaded.

/p only:

When only **/p** is used, Link loads all areas from all modules so that the areas have the following order in memory, starting at the specified program origin: all common, data, and program areas from the first module, followed by all common, data, and program areas from the second module, and so on for each module loaded.

/d and /p:

When both **/d** and **/p** are used, Link separates the program areas from the data and common areas so that the program areas have the following order in memory, starting at the specified program origin: all program areas from the first module, followed by all program areas from the second module, and so on for each module loaded.

The data and common areas have the following order in memory, starting at the specified data origin: all common and data areas from the first module, followed by all common and data areas from the second module, and so on for each module loaded.

1.5 $memry

$memry is a global symbol recognized by Link to have special significance. If Link loads a module containing the definition of a global with the name **$memry**, it will store in that symbol the address of the first available

byte of memory following the data area (top of the data area + 1).

The value loaded into **$memry** depends on the use of the **/d** and **/p** switches.

When neither **/d** nor **/p** is used, or when **/p** is used by itself, Link will set **$memry** to point to the first available byte following all common, data, and program areas loaded.

When **/d** is used by itself, or when **/d** and **/p** are used together, Link will set **$memry** to point to the first available byte following all common and data areas loaded.

1.6 Separating Data and Program Areas

As has been discussed, **/d** and **/p** can be used to separate the data and program areas. There is one important fact to keep in mind--the Fortran disk driver module in the FORLIB.REL relocatable library file uses **$memry** to allocate space for disk buffers (128 decimal bytes each) and file control blocks (33 decimal bytes each). So, in this case, placing the data area before the program area requires that enough space be left between the two areas for the buffers and the file control blocks. Failure to leave enough space will likely cause the program area to be overwritten at runtime.

1.7 Relocatable File Format

A relocatable file consists of a single bit stream divided into individual fields of data. An individual field is not aligned on a byte boundary, except as noted below. There are two fundamental types of fields, also called Link items: **absolute** and **relocatable.** The first bit of an item indicates the type of the item; 0 indicates that the next 8 bits in the stream are to be loaded into memory as a single byte, without change. A 1 indicates that the item is a relocatable item, and the value of the next 2 bits from the bit stream indicates the type of item:

00 this is a special item, and is further defined in following paragraphs.

01 this is a program relative item. Link will add the current program base to the following 16 bits and put the result into the next two bytes in the program area.

02 this is a data relative item. Link will add the current data base to the following 16 bits and put the result into the next two bytes in the data area.

03 this is a common relative item. Link will add the current common base to the following 16 bits and put the result into the next two bytes in the common area.

All 16-bit fields represent two-byte values; the first byte is the low-order byte.

A special item consists of the bit stream 100 (already discussed) followed by:

a four-bit control field;

an optional A field consisting of a two-bit address type which can have the same values and meanings as a relocatable item (described in the preceding paragraph), with the exception that 00 indicates that the next 16 bits are to be loaded as an absolute address;

an optional B field consisting of 3 bits that give the length in characters of the following symbol, and 8 bits for each character of the symbol.

The general form of a special item is:

100 xxxx <u>yy nn</u> <u>zzz + characters of the symbol name</u>
A field B field

where

100 is a 3-bit value;

xxxx is a 4-bit control field, defined below;

yy is a 2-bit address type field;

nn is a 16-bit value;

zzz is the 3-bit symbol length.

The following special items have a B field only:

0 entry symbol (definition of a global);
1 select COMMON block with following name;
2 program name;

3 request library search;
4 start of COBOL independent segment; the symbol name is 1 byte, which contains the binary number of the segment in the range 50-99 decimal; values in the range 0-49 are reserved for expansion.

The following special items have an A field and a B field:

5 define COMMON size;
6 chain external; the A field is the head of the address chain, the B field is the name of the external symbol;
7 define entry point; the A field is the address, the B field is the name.

The following special items have an A field only:

8 external - offset; used for JMP and CALL to externals;
9 external + offset; the A field value will be added to the two bytes starting at the current location counter immediately before execution;
10 define size of data area; the A field is the size;
11 set loading location counter to the value of the A field;
12 chain address; Link will replace all entries in the address chain with the current location counter; the value of the A field points to the head of the chain, and the last entry in the chain has an address field of binary 0;
13 define program size; the value of the A field is the size;
14 end module; forces to byte boundary.

The following special item has neither an A nor a B field:

15 end of relocatable file.

1.8 Hex File Format

A file created in the Intel Hex format consists of lines each of which must start with a colon, :. The rest of each line consists of ASCII characters, each of which represents one hexadecimal digit. The line is interpreted by reading two digits at a time to form bytes.

The first byte is a binary number which indicates the number of data bytes on the line. The next two bytes represent the memory address at which the data bytes on the line will be loaded. The first address byte is most significant. The byte following the address bytes is always zero. The rest of the bytes on the line, up to but not including the last one, are the data bytes to be loaded into memory. The final byte is the checksum. When all bytes (2-digit groupings) on the line are added with the checksum, the result must be zero, otherwise the line is incorrect in some way.

If the data byte count is zero, the two-byte address field is the start-of-execution address for the program being loaded. This address is not used in programs prepared for any Cromemco operating system, and the Cromemco Macro Assembler sets it to zero.

A file in hex format is not executable. The Read command in the Cromemco Debug program must be used to load a hex file into memory. Refer to the Cromemco Macro Assembler Manual, part number 023-0039, for the details concerning the use of Debug.

Example:

```
:1C020000221303EB221503C5E14E2346ED431703234E2346ED431903234E2346DE
:1C021C00ED431B032A1303111D03010200EDB02A1503010200EDB02A170301023E
:1C02380000EDB0ED4B1D03CB7920123E08B9F26C023E0AB9F275023E0EB9F26C13
:0000000000
```

1.9 Link Error Messages

Error messages preceded by % are cautionary only--Link is warning the user that something has occurred, or will occur, which might be undesirable. Error messages preceded by ? indicate that Link cannot perform a requested action or that something has happened during the loading of a module which has completely confused Link.

%2nd COMMON Larger /<name>/

The first definition of the COMMON block with the name <name> was not the largest definition. The module containing the largest definition of a particular COMMON block must be loaded before other modules defining that COMMON block.

?<filename> Not Found

Link cannot find a file with the name <filename>.rel.

?Can't Save Object File

Link cannot save the executable object file on disk. The disk directory is full, there is no available space on the disk, or an I/O error occurred during the save operation.

?Command Error — Link cannot recognize the command.

?Intersecting Data Area ,Start = xxxx
,Public = <symbol name> (xxxx)
,External = <symbol name> (xxxx)
or
?Intersecting Program Area ,Start = xxxx
,Public = <symbol name> (xxxx)
,External = <symbol name> (xxxx)

The program and data areas intersect and an address or external chain entry is in this intersection. During the loading process, Link must move the program and data areas around in memory, which requires conversion of addresses and external chain entries from what are called final values to current values and back. This message indicates that a final value cannot be converted to a current value since it is in the area intersection, and Link can't continue. Link displays the symbol name to tell the user in which module the problem occurred.

?Loading Error — The last input file is not a properly formatted relocatable file.

%Mult. Def. Global <name>

More than one definition for the global symbol <name> was found

during the loading session.

?No Start Address — **/g** was entered as a command, but no main program has been loaded.

Origin Above Loader Memory, Move Anyway (Y or N)?
or
Origin Below Loader Memory, Move Anyway (Y or N)?

Link displays this message following a **/e** or **/g** if either the data area or program area origin is addressed either below or above the memory range usable by Link. This range is 100H to the first byte before the start of the operating system. A **Y** response will cause Link to move the area and continue; any other response will cause Link to return control to the operating system.

Link creates and saves the new executable file (in response to the use of **/n**) before attempting to move the completely-loaded program and data areas around in memory, so the response to this question will not affect the newly created file.

?Out of Memory — There is not enough memory to finish loading the program.

%Overlaying Data Area

A **/d** will cause a previously-loaded data area to be destroyed.

%Overlaying Program Area

A **/p** will cause a previously-loaded program area to be destroyed.

?Start Symbol - <name> - Undefined

The symbol specified as a start address following a /e or /g is not defined.

SECTION 2

LIB

Lib is a program which allows the user to build, maintain, and examine libraries of relocatable object code modules. These libraries may then be searched by the Link program in response to the **/s** control switch to satisfy unresolved external symbol references. A library file consists of one or more relocatable object code modules which have been created by the compilation or assembly of individual source files which contain ENTRY statements defining global symbols. A valuable feature of this library capability is that the Link program loads only the containing module when satisfying an external reference; Link doesn't load modules containing only unreferenced entry points.

Lib contains command features which permit the concatenation of the modules in .REL files to form new libraries. Further, the Lib command syntax permits the specification of ranges of modules within .REL files. This powerful feature allows the extraction of multiple contiguous modules from existing libraries and facilitates the replacement of a single module within a library with an updated version.

2.1 Lib Commands

Lib is controlled by the use of commands which may be entered on the operating system command line that initiates execution or in lines entered in response to the Lib prompt, *. In either case, Lib responds to commands by adding additional modules to the library under construction or by listing information about an existing library file. A command has the form:

newname = filename<module specifier> ... command switch ...

A comma must separate each **filename<module specifier>** entry in a command.

newname = is optional.
Lib will create a new library file with the name **newname.rel** if the name is included in the command. Otherwise, Lib will use the default name FORLIB.REL.

While Lib is constructing the library, it will use the extension .LIB for the file. After the user types either the **/e** or **/r** switch, Lib will complete the file and change the

extension to .REL, unless the user has included a different extension in **newname.**

filename is the name of a file containing one or more relocatable object code modules. The file may be either an existing library file, or the .REL file output from the Cromemco Macro Assembler or any one of the C, Cobol, Fortran, or RPG II compilers.

A filename has the form: **drive:name.ext,** where **drive:** and **.ext** are optional.

drive: must be a single letter in the range A-H, followed by a colon; the default is the current drive.

name must be 1 - 8 characters in length. All ASCII characters are valid except the space, control characters, any of ? * , = / or the DEL character (7FH).

.ext must be 1 - 3 characters long, with the same restrictions as **name;** the default is .REL.

<module specifier>

is optional.
This clause specifies which modules in the file are to be included within the library or examined with the **/l** or **/u** switches. Lib will use all modules in the preceding file if the module specifier is omitted. The angle brackets, < >, are required.

A single module is specified by using the name of the module:

SIN.

Modules preceding or following the module with the given name are specified by using an offset of -255 to +255:

SIN+1 specifies the first module following the one named SIN,

SIN-1 specifies the first module preceding the one named SIN.

Ranges of modules may be specified by using

two dots:

..SIN — specifies all modules from the beginning of the file up to and including the one named SIN;

SIN.. — specifies all modules from and including the one named SIN to the end of the file;

SIN..COS — specifies all modules between the modules named SIN and COS, inclusive.

Ranges and relative offsets may be used in combination:

SIN+1..COS-1.

command switch

Descriptions of the available command switches follow in the next paragraph.

2.2 Lib Command Switches

This paragraph discusses the functions of the command switches available in Lib. Each switch must be preceded by a slash, **/**.

C — **/c** causes Lib to discard the library under construction and to start again.

E — **/e** causes Lib to complete the construction of the new library and **exit** to the operating system. Lib saves the new library file, changes the extension in the name of the temporary library file from .LIB to .REL, and deletes any existing library with the same name. This switch, or **/r**, must be used to save the new library on disk.

Note that **/e** always causes Lib to create and save a new library file, whether with the name **newname.rel** or with the default name, FORLIB.REL, so care must be taken not to lose an existing library with the same name.

H — **/h** changes the display radix used in the listing of symbol values to **hexadecimal.** This

is the default.

L **/l** causes Lib to **list** information about all modules in each file whose name precedes this switch in the command line. The information listed for each module includes the name, the program and data lengths, entry points with their relative values, and external references with their relative locations. Lib also lists a combined cross reference with the name, relative value, defining module name, and referencing module names of each symbol declared ENTRY in at least one of the files. The listing will appear on the terminal.

O **/o** changes the display radix to **octal.** Hexadecimal is the default.

R **/r** causes Lib to **rename** the temporary library file. **/r** performs the same functions as **/e** except that Lib will not return control to the operating system. This allows multiple libraries to be built or examined during one Lib session. This switch, or **/e**, must be used to save the new library on disk.

U **/u** will cause Lib to search all files whose file names precede this switch on the command line and list the symbols which would remain **undefined** after a search through those files in the specified order.

Any command specifying a list of module names causes Lib to concatenate those modules at the end of the library being constructed. This means that the following Lib command

```
*file1,file2 <mod1>, test
```

is equivalent to:

```
*file1
*file2 <mod1>
*test
```

A module containing code which refers to an ENTRY symbol in a second module must be loaded into the library before the second module. This is required because Link makes only a single pass through a library in response to the **/s** command switch. Link does not load modules

containing only unreferenced entry points, and it does not back up to rescan the library when external symbols are encountered later in its scan.

The **/u** command switch may be used to list the symbols which would remain undefined following a single pass through a library. If a module in the library contains a backward reference to a symbol in a preceding module, that symbol will be listed. Certain symbols in the standard Fortran and Cobol libraries will be listed as undefined, but this will not cause a problem when loading Fortran and Cobol programs--the modules containing these symbols are always loaded automatically by Link.

Lib can be used to list information about files without creating a new library file. Type **Lib** from the operating system prompt level, followed by the list of file names to examine, followed by **/l** or **/u.** Lib will examine those files, list the information, and return control to the operating system.

Examples:

The first five examples illustrate the use of Lib to make a library named **tranlib.rel.**

A.lib intitiates execution of Lib.

*tranlib=sin,cos,tangents<tan..atan>

causes Lib to create a temporary file named **tranlib.lib** and insert all modules found in the files **sin.rel** and **cos.rel,** and all modules in the file named **tangents.rel** from the one named **tan** to the one named **atan,** inclusive.

*exp causes Lib to add all modules found in the file **exp.rel** to the end of **tranlib.lib.**

*tranlib.lib/u

causes Lib to search for and list any global symbols that would remain undefined after one search pass through **tranlib.lib.**

*/e causes Lib to save the file **tranlib.lib,** change its name to **tranlib.rel,** and return control to the operating system.

A.lib newlib=coblib<..dskdrv-1>, newdrv, coblib<dskdrv+1..>/e

causes Lib to create a new file named **newlib.rel** from the file named **coblib.rel** by replacing the module named **dskdrv** with the module name **newdrv.** All other modules from **coblib.rel** will be included in the new file.

2.3 Lib Error Messages

?Can't enter file — There is no more room in the directory to make the entry for the .LIB temporary file.

?Command error — The typed command is invalid.

?File not found — The specified input file was not found on the specified disk.

?File read error — The specified input file cannot be read.

?First module in Until clause after last

The first module in the specified range of modules exists in the file following the second module, so this range specification is meaningless.

?Library write error

The new library cannot be saved on disk because there is not sufficient room.

?Module name/number not found in file

The specified module name does not exist within the selected file.

?Out of memory — A module exists that is too large to load into memory. The only solution to this error is to rewrite the module so that it occupies less memory.